

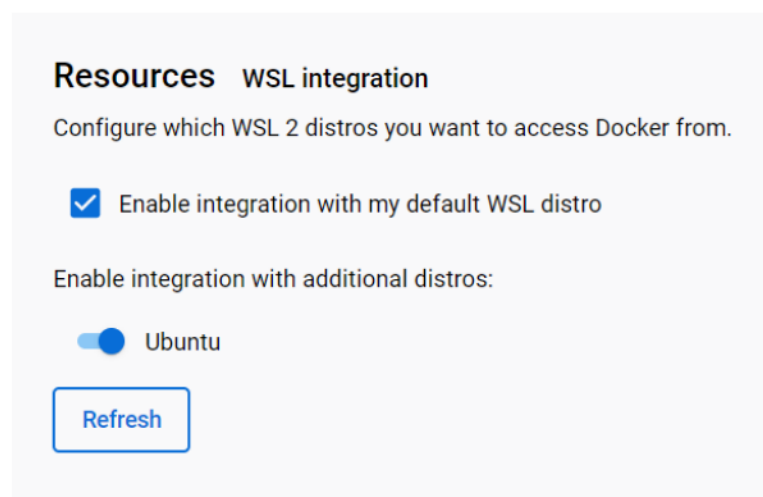
Разворачивание GitLab как локального приложения с помощью Docker. Настройка и проверка конвейера с помощью git. Аутентификация.

▼ Особенности WINDOWS

Windows 10. WSL должен быть установлен

Установите <https://docs.docker.com/docker-for-windows/wsl/#download>

Проверьте в Docker Desktop, что интеграция с Ubuntu включена:



Откройте терминал WSL и запустите `docker run hello-world` для проверки, что docker работает внутри WSL.



При вопросах проглядите <https://learn.microsoft.com/en-us/windows/wsl/tutorials/wsl-containers>

▼ Особенности Linux (нужна Ubuntu 22.04 LTS или 23.04)



Установите (при необходимости) и подготовьте систему, как описано тут:

<https://docs.docker.com/desktop/install/linux-install/>

Скачайте пакет установщика (см. <https://docs.docker.com/desktop/install/ubuntu/>):

▼ Если требуется обновить Ubuntu до подходящей версии

Требуется установить обновления всех компонентов, в т.ч. пользовательских

Далее можно запустить обновление самой системы с помощью `/usr/lib/ubuntu-release-upgrader/check-new-release-gtk`

У меня 16 октября обновление с 20.04 не сработало, пришлось устанавливать чистую LTS версию 22.04 совместимую с Docker.

Установите:

```
sudo apt-get update
sudo apt-get install ./docker-desktop-4.24.2-amd64.deb
```

Далее все команды должны работать как на нативной Ubuntu, так и на установленной через WSL:

Установить/проверить агент для работы с протоколом SSH

```
sudo apt install ssh
```

Установить/проверить агент для работы git репозиториями (при необходимости)

Установить/проверить docker

```
#может понадобиться удалить некоторые неофициальные относящиеся к docker пакеты
for pkg in docker.io docker-doc docker-compose docker-compose-v2 podman-docker containerd runc; do sudo apt-get remove $pkg; done
```

```
sudo apt install docker
```

▼ Создать локальный проект вашего отчёта

Создайте папку, где будет всё, связанное с проектом

```
mkdir leading_engineering_schools
cd leading_engineering_schools
```

Поместите этот код в docker-compose.yml:

```
version: '3.6'
services:
  web:
    image: 'gitlab/gitlab-ce:latest'
    restart: always
    environment:
      GITLAB_OMNIBUS_CONFIG: |
        external_url 'http://172.17.0.1:80'
        # Add any other gitlab.rb configuration here, each on its own line
    ports:
      - '80:80'
      - '443:443'
      - '22:22'
    volumes:
      - '$GITLAB_HOME/config:/etc/gitlab'
      - '$GITLAB_HOME/logs:/var/log/gitlab'
      - '$GITLAB_HOME/data:/var/opt/gitlab'
    shm_size: '256m'
  runner:
    image: 'gitlab/gitlab-runner:latest'
    volumes:
      - '/var/run/docker.sock:/var/run/docker.sock'
      - '$GITLAB_HOME/runner-config:/etc/gitlab-runner'
```



если вы работаете не внутри WSL может понадобиться изменить строку:
`external_url 'http://127.0.0.1:80'`

Запустите две команды для сборки и запуска контейнеров с приложением:

```
export GITLAB_HOME=/home/adam/devops/
docker compose up -d
```

... и добейтесь, чтобы контейнеры запустились.

После 3 минут можно начинать проверять, после 5 можно смотреть `docker ps` и проверять статус запуска — нет ли ошибок.

Далее можно подключаться к <http://localhost:80> (пользователь `root` , про пароль см. в конце этого текста) и настраивать доступ к только что развёрнутому приложению GitLab (в интерфейсе нужно создать token для `glab`) и к его репозиторию (через `glab` или web интерфейс развёрнутого приложения скопируйте ваш публичный `id_rsa.pub` ключ из папки `~/.ssh/` в настройки приложения GitLab):

▼ Установить `glab` для работы через командную строку с GitLab

```
sudo snap install glab
```

```
glab auth login
```

выбрать ? `GitLab Self-hosted Instance`

```
#Ubuntu 22.04 LTS
glab auth login
```

```
gitlab.com
> GitLab Self-hosted Instance
```

```
? GitLab hostname: localhost
? API hostname: [? for help] (localhost) #тут просто ВВОД
```

```
? How would you like to login? Token
```

Теперь получим token доступа http://localhost/-/profile/personal_access_tokens

User Settings > Access Tokens

Search page

Personal Access Tokens

You can generate a personal access token for each application you use that needs access to the GitLab API. You can also use personal access tokens to authenticate against Git over HTTP. They are the only accepted password when you have Two-Factor Authentication (2FA) enabled.


Active personal access tokens 0

Add new token

Token name	Scopes	Created	Last Used ?	Expires	Action
This user has no active personal access tokens.					

Personal Access Tokens

You can generate a personal access token for each application you use that needs access to the GitLab API. You can also use personal access tokens to authenticate against Git over HTTP. They are the only accepted password when you have Two-Factor Authentication (2FA) enabled.

Active personal access tokens  0

Add a personal access token

Token name

For example, the application using the token or the purpose of the token.

Expiration date



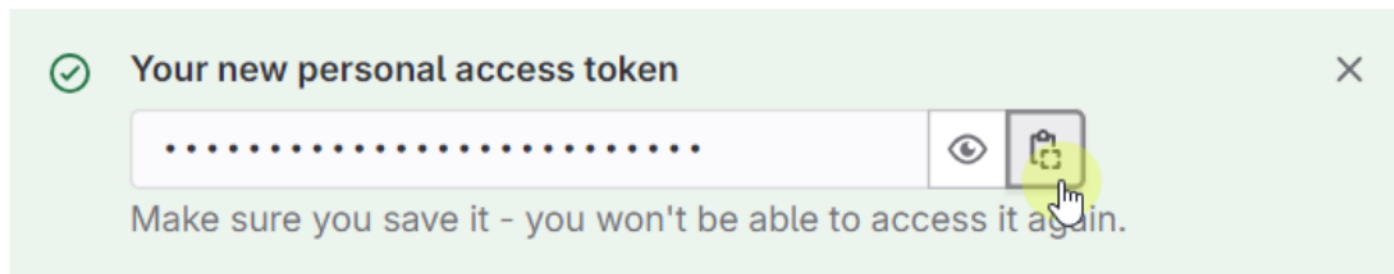
Select scopes

Scopes set the permission levels granted to the token. [Learn more.](#)

- ☒ **api**
Grants complete read/write access to the API, including all groups and projects, the container registry, and the package registry.
- ☒ **read_api**
Grants read access to the API, including all groups and projects, the container registry, and the package registry.
- ☒ **read_user**
Grants read-only access to the authenticated user's profile through the /user API endpoint, which includes username, public email, and full name. Also grants access to read-only API endpoints under /users.
- ☒ **create_runner**
Grants create access to the runners.
- ☒ **k8s_proxy**
Grants permission to perform Kubernetes API calls using the agent for Kubernetes.
- ☒ **read_repository**
Grants read-only access to repositories on private projects using Git-over-HTTP or the Repository Files API.
- ☒ **write_repository**
Grants read-write access to repositories on private projects using Git-over-HTTP (not using the API).
- ☒ **read_registry**
Grants read-only access to container registry images on private projects.
- ☒ **write_registry**
Grants write access to container registry images on private projects.

Create personal access token

Cancel



Далее необходимо авторизовать с помощью этого токена ваш локальный (удалённый с т.з. gitlab) клиент glab

```
glab auth login
```

Теперь мы можем подключаться к gitlab.com с помощью утилиты `glab`

▼ Настроить ssh ключ для доступа к репозиторию

```
sudo apt install ssh
ssh-keygen -t rsa
sudo chmod 400 ~/.ssh/id_rsa.pub
```

Добавьте ваш ключ для SSH в настройках GitLab, чтобы иметь доступ к репозиторию. С помощью утилиты `glab` создайте удалённый репозиторий в созданном GitLab приложении, а в нём простейший файл описания конвейера `.gitlab-ci.yml`, который бы создавал в качестве артефакта простой текстовый файл `file.txt` с любым вашим текстом (он будет создаваться в окружении, которым управляет ранер).

Теперь нужно добавить ранер (он поднят в отдельном контейнере, но в интерфейсе GitLab нужно найти, как его привязать — см. особенности в конце задания).

Осуществите `push` файла `.gitlab-ci.yml` в репозиторий, и проверьте, что конвейер успешно завершается.

Как только это произойдёт, задание можно считать выполненным. Но ещё нужно опубликовать главные появившиеся в процессе работы файлы и папки вашего проекта. Вот их список:

```
./docker-compose.yml
./runner-config/
./logs/
```

Эти элементы вашей рабочей папки проекта отчётности загрузите в ваш уже личный репозиторий на стандартном всем доступном gitlab.com, предварительно там зарегистрировавшись. Проверьте, что доступ на чтение к этому репозиторию есть к нему у всех и в качестве результата предоставьте ссылку на него.

В качестве отчёта приложите видео компьютера со стороны, на котором вы запускаете контейнер с GitLab, который работает по адресу `http://localhost`, в котором вы создаёте репозиторий с помощью `glab`, который вы локально клонируете (`git clone`) а затем локально же вносите изменения, осуществляете `push` на локальный GitLab, после чего пайплайн успешно запускается, завершается, и вы демонстрируете, как к созданному артефакту вы можете получить доступ из того же окружения, в котором вы вносили изменения в локальную копию репозитория.



Особенности работы

1. Адресом GitLab необходимо указать 172.17.0.1: это способ изнутри контейнера, обратиться к localhost окружения, в котором работает контейнер
2. Пароль пользователя `root` установленного GitLab находится в контейнере в файле `/etc/gitlab/initial_root_password`. Поскольку наш `docker-compose.yml` перенаправляет эту папку контейнера на папку `$GITLAB_HOME/config/` нашего окружения, можно прямо из рабочего окружения посмотреть содержание файла, например, так: `cat $GITLAB_HOME/config/initial_root_password`
3. При установке раннера GitLab выдаст код для его регистрации. Но поскольку ранер работает в контейнере, то команду нужно выполнить внутри контейнера ранера. Зайти в командную строку в контейнере можно с помощью `docker exec -it <runner container name> bash`
4. Чтобы репозиторий кода нас узнал, когда мы будем с ним работать, нужно сгенерировать файл `id_rsa.pub` в папке `~/.ssh` запустив `ssh-keygen -t rsa` в папке `~/.ssh` основного окружения и содержимое этого файла добавить в ключи пользователя нашего локального (`http://localhost`) GitLab.
5. Также необходимо указать настройки пользователя для git:

```
git --global user.name <My Name>
git --global user.email <my@ema.il>
```